// **Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

void loadCourseDataFromFile(String csvPath, HashTable\* hashTable) {

print(“Loading CSV file: “ + csvPath)

// Initialize the CSV parser using the given path

csvParser file = csvParser(csvPath)

vector<String> header = file.getHead()

for each column in header {

print(column + “ | “)

}

print(“”)

try {

// Loop through each row in the CSV file

for (unsigned int I = 0; I < file.rowCount(); i++) {

// **Validate that each row has at least two parameters (courseNumber and courseName)**

if (file[i].size() < 2) {

print(“Error: Invalid format.”)

continue

}

String courseNumber = file[i][0]

String courseName = file[i][1]

Course course = new Course(courseNumber, courseName)

**// Check for prerequisites and ensure they exist as valid courses**

for (int j = 2; j < file[i].size(); j++) }

String prerequisite = file[i][j]

if (!isCourseInFile(prerequisite, file)) {

print(“Error: prerequisite “ + prerequisite + “ does not exist in the file.” )

} else {

course.addPrerequisite(prerequisite)

}}

hashTable->Inser(course)

}

} catch (csv::Error &e) {

print(e.what())

}}

**// Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

structure Course {

String courseNumber

String courseName

List<String> prerequisites

// constructor

Course(String number, String name) {

courseNumber = number

courseName = name

prerequisites = new List<String>( ) // Initalize empty list of

}

// Method to add a prerequisite to the course

void addPrerequisite(String prerequisite) {

prerequistites.add(prerequisite)

} }

// Define the Node structure for the hash Table

struct Node {

Course course // Stores course information

unsigned int key // Stores the hashed key

Node\* next // Pointer to handle chaining in case of collisions

// Contructor to initialize the node

node(Course courseData, unsigned int hashKey) {

course = courseData

key = hashkey

next = nullptr

} }

// Define the Hash table class

class HashTable {

private:

vector<Node> table

unsigned int tableSize = 179

unsigned int hash(String courseNumber) {

unsigned int hashValue = 0

for each char in courseNumber {

hashValue += char

}

return hashValue % tableSize

}

public:

HashTable() {

table.resize(tableSize)

}

// Method to insert a course into the hash table

void Insert(Course course) {

unsigned int hashKey = hash(course.courseNumber)

Node\* newNode = new Node(course, hashKey)

// If the slot is empty, add the node

if (table[hashKey].next == nullptr) {

table[hashKey].next = newNode

} else {

// Handle collision using chaining

Node\* temp = table[hashKey].next

while (temp->next != nullptr) {

termp = temp->next

}

temp->next = newNode

} }

**// Design pseudocode that will print out course information and prerequisites**

void printCourseInfo(HashTable\* hashTable, String courseNumber)

// Search for the course in the hash table

Course\* course = hashTable->Search(courseNumber)

if (course != nullptr) {

// Print course information

print(“Course Number: “ + course->courseNumber)

print(“Course Name: “ + course->courseName)

// Print prerequisites if available

if (course->prerequisites.size() > 0) {

print(“Prerequisites: “)

for each prerequisites in course->prerequisites {

print(“ – “ + prerquisite)

}

} else {

print(“No Prerequisites”)

}

} else {

print(“Course not found.”)

} }

void printAllCourses(HashTable\* hashTable) {

for (unsigned int i = 0; I < hashTable->table.size(); i++)

Node\* temp = hashTable->table[i].next

while (temp != nullptr) {

print(“Course Number: “ + temp->course.courseNumber)

print(“Course Name: “ + temp->course.courseName)

if (temp->course.prerequisites.size() > 0) {

print(“Prerequisites: “)

for each prerequisite in temp->course.prerequisites {

print(“ – “ + prerequisite)

}

} else {

print(“No prerequisites”)

}

**// Create pseudocode for a menu**

void displayMenu() {

println(“Menu: “)

println(“Option 1: Load the file data into the data structure.”)

println(“Option 2: Print an alphanumerically ordered list of all the courses in the Computer Science department.”)

println(“Option 3: Print the course title and the prerequisites for any individual course.”)

println(“Option 9: Exit the program.”)

}

void main() {

HashTable\* hashTable = newHashTable()

int choice = 0

while choice != 9 {

displayMenu()

print(“Enter your choice: “)

choice = user input

switch choice {

case 1:

loadCoursedataFromFile(filePath, hashTable)

break

case 2:

if hashTable->isEmpty() {

print(“No data in course”)

} else {

printSortedCourses(hashTable

}

break

case 3:

if hashTable->isEmpty() {

print(“No data in course”)

} else {

print(“Enter course number: “)

String courseNumber = get user input

printCourseInfo(hashTable, courseNumber)

}

break

case 9:

print(“Exit programing”)

break

default:

print(“Invalid input”)

} } }

void printSortedCourses(HashTable\* hashTable) {

Vector<Course> sortedCourses

// Traverse the hash table and collect all courses

for (unsigned int i = 0; i < hashTable->table.size(); i++)

Node\* temp = hashTable->table[i].next

while (temp != nullptr) {

sortedCourses.add(temp->course)

temp = temp->next

} }

// Sort the courses by alphanumeric course number

sortCourses(sortedCourses)

// Print the sorted courses

for each course in sortedCourses {

print(“Course Number: “ + course.courseNumber + “ | Course Title: “ + course.courseName)

} }

void sortCourses(Vector<Course> courses) {

// used a bubble sort for alphanumerical order

for i = 0 to courses.size() – 1{

for j = 0 to courses.size() – i – 1{

if courses[j].courseNumber > courses[j + 1].courseNumber {

swap courses[j] with courses[j + 1]

} } } }